Volume 2, Issue 3, March 2012 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Enhanced LZW (Lempel-Ziv-Welch) Algorithm by Binary Search with Multiple Dictionary to Reduce Time Complexity for Dictionary Creation in Encoding and Decoding

*Nishad PM
Ph.D Scholar, Department Of Computer Science
NGM College, Pollachi, India
nishadpalakka@yahoo.co.in

**R.Manicka Chezian
Associate prof., Department of computer science
NGM College Pollachi, Coimbatore, India

***Abstract:** The LZW is a universal lossless data compression algorithm which takes linear time in encoding and decoding. This paper proposes a new methodology to reduce time complexity by complaining binary search with insertion point and multiple dictionaries. The time taken for shifting before insertion gradually reduced and also the time taken for searching in the dictionary for pattern in encoding and decoding is reduced. Therefore the enhanced LZW by multiple dictionaries with the binary search and insertion point reduces the time complexity. The proposed methodology may be the best for the communication and minimizes the time complexity in pattern identification for compression and decompression. The proposed methodology reduces the time complexity with the binary search tree (BST) and with binary search. The experimental result shows 99 percentage improvements on compression and decompression.*
***Keywords:** Lossless Compression, LZW, Binary Search, Compression, Decompression, Compression Ratio, Encoding, Decoding, BST and Multiple dictionaries.*

Introduction

Data compression is a method of encoding rules that allows substantial reduction in the total number of bits to store or transmit a file. Two basic classes of data compression are applied in different areas currently that are lossy and lossless compression [l, 2]. One of the lossless data compression widely used is LZW data compression, it is a dictionary based algorithm. LZW compression is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch [3]. Lempel-Ziv-Welch (LZW) [3] this algorithm proposed by Welch in 1984. LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. LZW compression is fast comparing to other algorithms. This algorithm is an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 (LZ78) [4]. The first algorithm of Lempel and Zive was published in 1977 and it is named as LZ77 [5]. The LZ77 [5] and LZ78 [4] are otherwise called LZ1 and LZ2 respectively like The LZW algorithm uses dictionary and index for encoding and decoding operation. It creates a dictionary and if a mach is found in the dictionary then corresponding string is replaced by the index. In the paper [12] proposed binary search with insertion point with a sorted table to reduce the time complexity. There are several algorithms like DEFLATE and GZIP uses the LZ family algorithms. LZW compression became the first widely used universal data compression method on computers. After the invention of LZW there are lots of improvements and enhancement done in LZW for data compression that is discussed in this section. LZW compression works best for files containing lots of repetitive data especially for text and monochrome images.

The LZW algorithm uses dictionary while decoding and encoding but the time taken for creating the dictionary is large, so to reduce the time complexity a new methodology is proposed in this paper. The number of shift before of a new pattern and the number of comparison required to find the pattern in the dictionary is reduced after the implementation of multiple dictionaries. The experimental result shows massive reduction in the time complexity.

LZW [3] compression uses a code table [6] common choice is to provide 4096 entries in the table. In this case, the LZW encoded data consists of 12 bit codes, each referring to one of the entries in the code table. Decompression is achieved by taking each code from the compressed file, and translating it through the code table to find what character or characters it represents. Codes 0-255 in the code table are always assigned to represent single byte from the input file. When the LZW program starts to encode a file, the code table contains only the first 256 entries, with the remainder of the table being blank. This means that the first code in the compressed file is of single byte from the input file being

converted to 12 bits. As the encoding continues, the LZW algorithm identifies repeated sequences in the data, and adds them to the code table. Compression starts the second time a sequence is encountered. The key point is that a sequence from the input file is not added to the code table until it has already been placed in the compressed file as individual characters (codes 0 to 255). This is important because it allows the decompression program to reconstruct the code table directly from the compressed data, without having to transmit the code table separately.

The compression algorithm uses two variables: *CHAR* and *STRING*. The variable, *CHAR*, holds a single character, (i.e.), a single byte value between 0 and 255. The variable, *STRING*, is a variable length string, (i.e.), a group of one or more characters, with each character being a single byte. The algorithm starts by taking the first byte from the input file, and placing it in the variable, *STRING*. Table -1 show this action in line 1. This is followed by the algorithm looping for each additional byte in the input file. Each time a byte is read from the input file it is stored in the variable, *CHAR*. The data table is then searched to determine if the concatenation of the two variables, *STRING+CHAR*, has already been assigned a code. If a match in the code table is not found, three actions are taken, (i), output the code for STRING, When a match in the code table is found, (ii), the concatenation of *STRING+CHAR* is stored in the variable, *STRING*, without any other action taking place. That is, if a matching sequence is found in the table, no action should be taken before determining whether there is a longer matching sequence is present in the table or not. An example of this is shown in line 5, where the sequence: *STRING+CHAR = 'AB'*, is identified as already having a code in the table. In line 6, the next character from the input file, '*B*', is added to the sequence, and the code table is searched for: '*ABB*'. Since this longer sequence is not in the table, the algorithm adds it to the table, outputs the code for the shorter sequence that is in the table (code 256), and starts over searching for sequences beginning with the character, *'B'*. This flow of events is continued until there are no more characters in the input file. The program is wrapped up with the code corresponding to the current value of *STRING* being written to the compressed file. LWZ compression algorithm is illustrated in Table-1. The Decompression algorithm uses four variables NCODE, OCODE, STRING, and CHAR. The decompression algorithm starts by taking the first byte from the input file and placing it in the variable, OCODE and output the OCODE. This action is shown in table-2 line 1. This is followed by the algorithm looping for each additional byte in the input file; each time a byte is read from the input file it is stored in the variable, NCODE. The data table is then searched to find the variable NCODE. If a match in the code table is not found STRING = OCODE +CHAR else if the NCODE is found then STRING = NCODE, then output the STRING. First Character of STRING is assigned to CHAR, then adds entry (OCODE+ CHAR) in table for and assigns NCODE to OCODE. This process will continue up to the last input. The decoding algorithm is shown in table-2.

The Existing implimentation of LZW, with BST and simple binary search has several limitations that directly leads to the time complexity. in LZW the comparison ratio riquired for the new pattern while encoding (NCODE) and decoding(OLDCODE+CHAR) is huge. For example if NCODE and OLDCODE+CHAR is 'ABBBD' then the absence of the pattern is returended after comparing all the elements in the dictionary (shown in table 1 line number -16 for encoding and table-2 line number 10 for decoding). in binary search tree Implimentation (BST) of LZW the search for the pattern 'ABBBD' the comparison required through 'AB','BC','ABB','BA', 'ABBB' then only the additional node is updated in the tree shown in figure-1. in simple binary seearch the Comparison ratio and Shifting before the insertion in huge shown in figure-2 that directly leads to tme complexity.

TABLE -1

LZW example: This shows the compression of the phrase: *ABCABBABBBCABABBBD*

S no	Char	String +Char	In Table?	Out put	Add to Table	New String	Commends
1	A	A				A	first character- no action
2	B	AB	No	A	256 =AB	B	
3	C	BC	No	B	257 =BC	C	
4	A	CA	No	C	258 =CA	A	
5	B	AB	Yes(256)			AB	first match found
6	B	ABB	No	256	259=ABB	B	
7	A	BA	No	B	260 =BA	A	
8	B	AB	Yes(256)			AB	Matches
9	B	ABB	Yes (259)			ABB	Longest Match
10	B	ABBB	No	259	261=ABBB	B	Longest Match index Substituted
11	C	BC	Yes(257)			BC	Matches
12	A	BCA	No	257	262=BCA	A	
13	B	AB	Yes(256)			AB	Matches
14	A	ABA	No	256	263=ABA	A	
15	B	AB	Yes(256)			AB	Matches
16	B	ABB	Yes(259)			ABB	Matches
17	B	ABBB	Yes(261)			ABBB	Longest Match
18	D	ABBBD	No	261	264=ABBBD	D	Longest Match index Substituted
19	EOF	D	No	D			End of file, Output string.

© 2012, IJARCSSE All Rights Reserved

TABLE -2 LZW examples: This shows the decompression of the phrase: *A B C 256 B 259 257 256 261 D*

S No	NCODE	OCODE	CHAR	STRING/ Output	New table entry
1	A	A	A	A	
2	B	A	B	B	256=AB
3	C	B	C	C	257=BC
4	256	C	A	AB	258=CA
5	B	256	B	B	259=ABB
6	259	B	A	ABB	260=BA
7	257	259	B	BC	261=ABBB
8	256	257	A	AB	262=BCA
9	261	256	A	ABBB	263=ABA
10	D	261	D	D	264= ABBBD

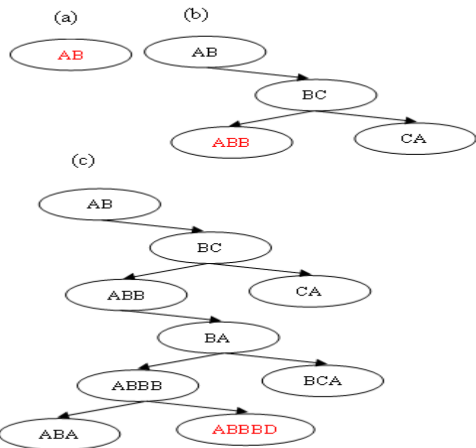


Figure -1 BST insertion for LZW encoding and decoding
(a) After insertion of 'AB'. (b) After insertion of 'ABB'.
(c) After insertion of 'ABBBD'.

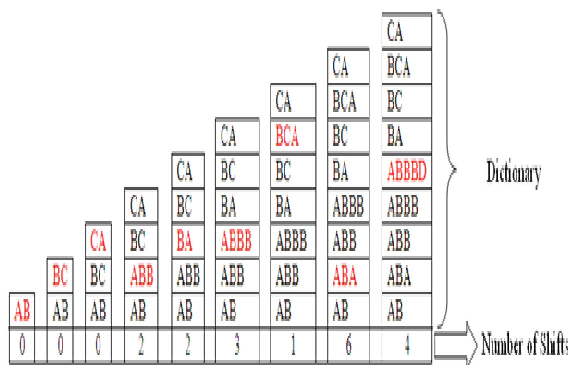


Figure - 2 LZW with Binary search with Sorted Dictionary with Number of Shifts before each insertion

Proposed approach

In the proposed methodology, the single dictionary is replaced with the multiple dictionaries. Each dictionary is unique. The approach reduces the time complexity by reducing comparison ratio and the shift ratio. The search in-between the dictionaries are switched based on the length of pattern in encoding as well as decoding. Each dictionary is constructed in sorted manner. To find the presents of pattern the simple binary search is implemented.

Encoding Algorithm

The proposed algorithm has two phases i) Switching and coding phase and ii) searching and updating phase. For encoding initially the initial dictionary or dictionary –zero contains all the possible roots in ascending order (0 to 255), so no insertion is made in the initial dictionary while decoding and encoding (compression). The first phase of the encoding algorithm uses three variables STRING, CHAR and DICT_CHOSE and also single dictionary is replaced with multiple dictionaries. The DICT_CHOSE has an integer value to identify the dictionary to be searched based on the pattern "STRING+CHAR", CHAR variable holds only a single character. The variable STRING is a variable length string (i.e.) it may have a group of one or more character with each character being a single byte. Initially the encoding algorithm initializes the variable DICT_CHOSE is 2 and then taking the first byte from the input file, and placing it in the variable, *STRING*. This is followed by the algorithm looping for each additional byte in the input file. Each time a byte is read from the input file it is stored in the variable, *CHAR*. Each time after storing byte to CHAR the switching is processed based on the DICT_CHOSE to determine if the concatenation of the two variables, *STRING+CHAR*, has already been assigned a code for example initially DICT_CHOSE is 2 and STRING is 'A' and CHAR is 'B' so the search for the pattern STRING+CHAR is made only within the dictionary -1 and the comparison required is zero (shown in figure -3) and the number of sift required is also zero. If the pattern STRING+CHAR is 'ABBB' the number of comparison required is zero and the number of shift also zero (shown in figure -3) but the algorithm use single dictionary the comparison required using single dictionary (BS) is 2 and using the simple LZW is 5 is shown in the table-3 And the number of shifts require for the insertion is 3 is shown in the figure-2. So using the multiple dictionaries and the binary search the number of comparisons and the number of shift is reduced so this directly leads to reduce the time complexity. If the pattern STRING+CHAR search result is true then the STRING+CHAR is store in to STRING and the DICT_CHOSE variable is incremented by one else CHAR is stored in to the STRING and the DICT_CHOSE is reset to two and the looping is continued.

For the searching and updating the dictionary the second phase of the algorithm is used each time after

© 2012, IJARCSSE All Rights Reserved

switching the dictionary is selected for the search base on the DICT_CHOSE. Binary search is used for searching if the pattern STRING+CHAR is reported in the corresponding dictionary then the function return true else another one variable updated is called 'COMPARE' by evaluating the condition if STRING+CHAR is less than key at index MID, if yes the COMPARE variable is set in to -1 else 1. So if the Matches is not found for the STRING+CHAR in the table (dictionary) then before exiting the binary search function MID is repositioned, based on the COMPARE to find the exact insertion position for the STRING+ CHAR (i.e.), if the compare value of COMPARE is 1 then the MID value is incremented by 1 and the MID is the insertion point for the STRING+CHAR and the STRING+CHAR is added to the position of MID to the corresponding dictionary before adding shifting is taken place.

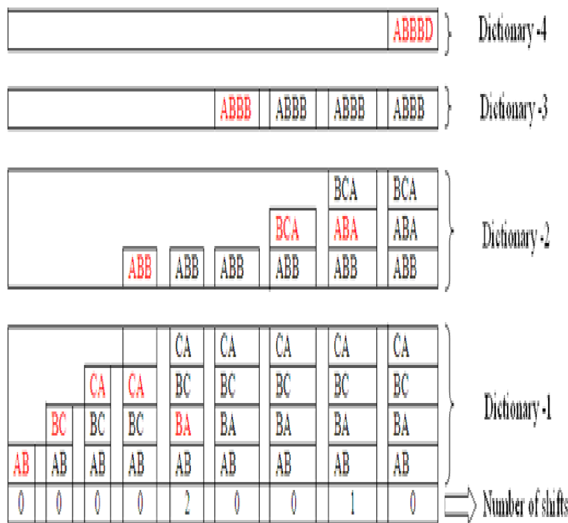


Figure-3 LZW Multiple sorted dictionaries and number of shifts before insertion in each dictionary

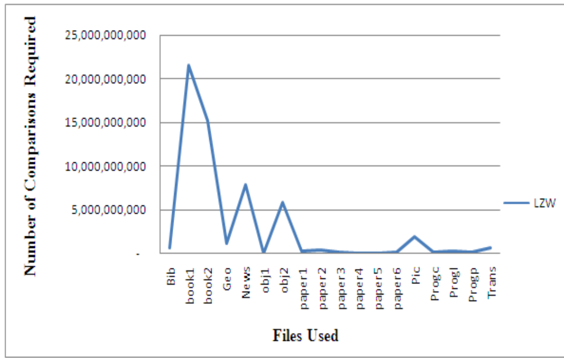
Table-3 Number of comparisons required for the patterns

S No	Pattern	LZW	BST	BS	MDBS
1	AB	0	0	0	0
2	BC	1	1	1	1
3	CA	2	2	2	2
4	AB	1	1	2	2
5	ABB	3	2	2	0
6	BA	4	3	2	2
7	AB	1	1	2	2
8	ABB	4	3	1	1
9	ABBB	5	4	2	0
10	BC	2	2	2	2
11	BCA	6	4	3	1
12	AB	1	1	2	2
13	ABA	7	5	3	1
14	AB	1	1	3	2
15	ABB	4	3	3	2
16	ABBB	6	5	1	1
17	ABBBD	8	5	3	0
TOTEL		56	43	34	21

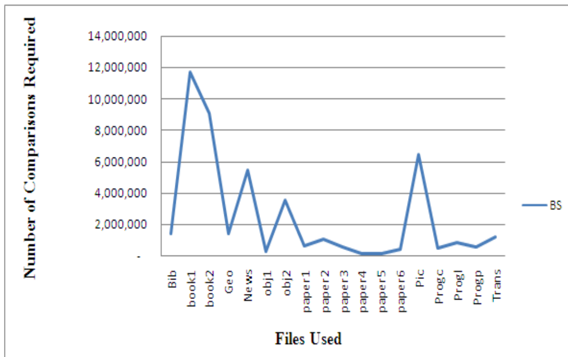
Decoding algorithm

The proposed decoding algorithm has also carries two phases i) switching and decoding phase and ii) searching and dictionary updating phase. The Decompression algorithm uses five variables DICT_CHOSE, NCODE, OCODE, STRING, and CHAR. The decompression algorithm starts by taking the first byte from the input file and placing it in the variable, OCODE and output the OCODE and initially the DICT_CHOSE variable set to two. This is followed by the algorithm looping for each additional byte in the input file; each time a byte is read from the input file it is stored in the variable, NCODE. The data table is then searched to find the variable NCODE. If a match in the code table is not found STRING = OCODE + CHAR else if the NCODE is found then STRING = NCODE and the DICT_CHOSE is incremented, then output the STRING. First Character of STRING is assigned to CHAR, then adds entry (OCODE+ CHAR) in table for and assigns NCODE to OCODE and DICT_CHOSE is reset to two. This process will continue up to the last input. The second phase is searching phase that is as same as the encoding algorithm.

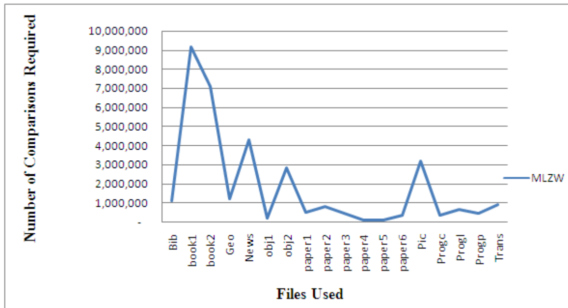
© 2012, IJARCSSE All Rights Reserved



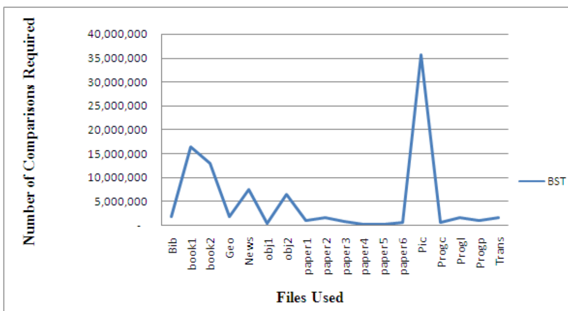
Graph -1 Comparisons Required for LZW for dictionary creation while encoding and decoding



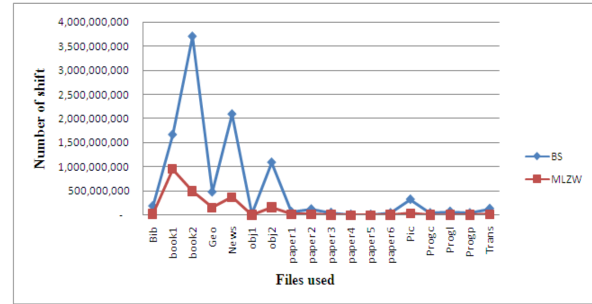
Graph -2 Comparisons Required for LZW using Binary Search (BS) for dictionary creation while encoding and decoding



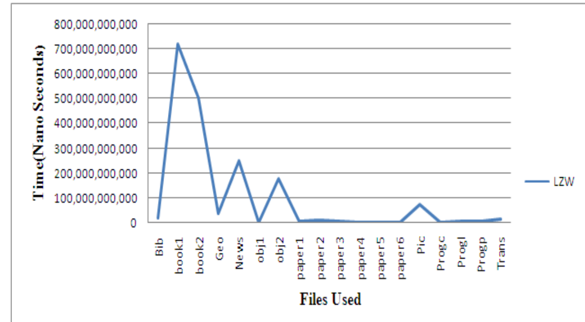
Graph -4 Comparisons Required for LZW using Multiple Dictionary (MLZW) for dictionary creation while encoding and decoding



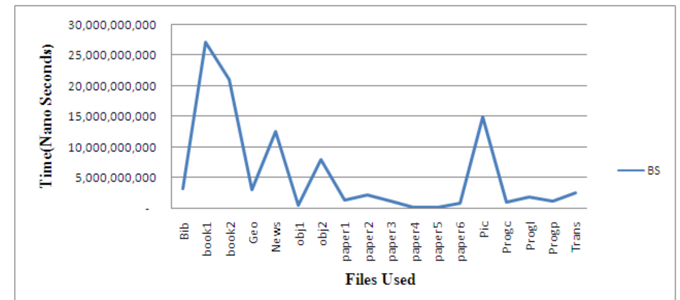
Graph -3 Comparisons Required for LZW using Binary Search Tree (BST) for dictionary creation while encoding and decoding



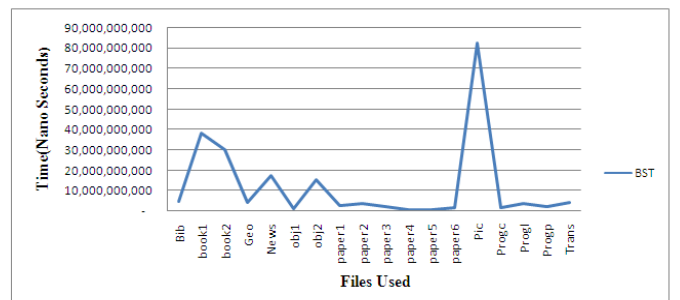
Graph -5 Number of shift require before inserting a new pattern in the dictionary for LZW using Binary Search and LZW using Multiple Dictionary (MLZW)



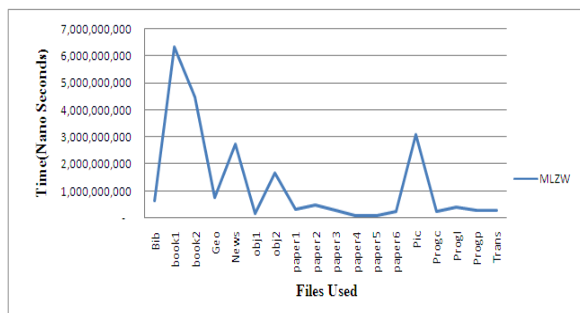
Graph -6 Average time taken for Encoding and Decoding using LZW in nanoseconds



Graph -7 Average time taken for Encoding and Decoding LZW using Binary Search in nanoseconds



Graph -8 Average time taken for Encoding and Decoding LZW using Binary Search tree (BST) in nanoseconds



Graph -9 Average time taken for Encoding and Decoding LZW using Multiple dictionaries (MLZW) in nanoseconds

In this proposed approach the time complexity in encoding and decoding is achieved by reducing the number of comparisons and number of shifts. the number of comparison required for the LZW is shown in graph-1, for Simple Binary search used by LZW is shown in graph-2, graph-3 shows the number of comparison required for dictionary creation LZW with Binary search tree and finally the graph -4 represent the LZW using binary search and Multiple dictionaries (MLZW) 99.94072 % of comparisons is reduced with LZW. Not only in the comparisons the Number of shift required also reduced in creating dictionary while encoding and decoding that is shown in the Graph-5. While comparing to the Binary search using Single dictionary the proposed approach reduces 77.32765132 % of Shift in creating the dictionaries. The average time taken for the encoding and decoding is shown in the Graph-6, graph-7, graph-8, and graph-9. The experimental result shows 99 % reduction of time complexity on compression and decompression.

Conclusion

LZW is a universal lossless data compression algorithm, which takes linear time in encoding. In this paper the problem of combining and summarizing LZW algorithm with binary search and multiple dictionaries data compression reduces the complexity of time at the maximum is analyzed. This is achieved by reducing number of comparisons and the number of shifts while creating the dictionary while encoding and decoding. The main focus of this paper in creating dictionary using LZW algorithm of any file with binary search and multiple dictionaries comprises the efficiency of compression time and decompression time efficiently. The experimental result shows average 99 % improvement in compressing and decompression, comparing to other already existing algorithm, procedures and methods.

The proposed work can be further enhanced and expanded for the authentication of compression and decompression techniques to obtain optimum accuracy in time.

ACKNOWLEDGMENT:

Dr. N. Nalayini. Associate Professor, Department of computer science, NGM College, Pollachi.

References:

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, "Text Compression English word" C1iffs:N. J. Prentice-Hall, 1990.
[2] Rafael C. Gonzalez and Richard E. Woods, "Digital Image Processing, Reading, Massachusetts": Addison-Welsley Publishing Company, 1992.
[3] WELCH,T. A. 1984." A technique for high-performance data compression". IEEE Comput. 17, 6, 8–19. 9
[4] ZIV, J. AND LEMPEL, A. 1978. "Compression of individual sequences via variable-rate coding". IEEE Trans. Inform. Theory 24, 5, 530–536.
[5] ZIV, J. AND LEMPEL, A. 1977. A "universal algorithm for sequential data compression". IEEE Trans. Inform. Theory 23, 3, 337–343.
[6] Steven W. Smith "The Scientist and Engineer's Guide to Digital Signal Processing " 1997
[7] Rahul Gupta, Ashutosh Gupta and Suneeta Agarwal "A Novel data compression algorithm for Dynamic data" IEEE REGION 8 SIBIRCON 2008 (266- 271)
[8] Rahul Gupta , Ashutosh Gupta and Suneeta Agarwal "A Novel Approach of Data Compression for Dynamic Data" 2008 IEEE
[9] Hidetoshi Yokoo - "Improved Variations Relating the Ziv – Lempel and Welch-Type Algorithms for Sequential Data Compression" IEEE Transactions on Information Theory, VOL. 38, NO. 1, JANUARY 1992 page (73-81)
[10] Kinsner and R H. Greenfield "The LEMPEL-ZIV-WEL (CLHzw) Data Compression Algorithm for Packet Radio" –IEEE
[11] Tinku Acharya and Joseph F "Enhancing Lempel-Ziv codes using an On-line Variable Length Binary Encoding" IEEE 1996
[12] NISHAD PM and Dr. N. NALAYINI "Enhanced LZW (Lempel-Ziv-Welch) Algorithm with Binary Search to Reduce Time Complexity for Dictionary Creation in Encoding and Decoding" published Proceedings of international conference ICICIC 2012-PSG tech 5- January 2012.

Biography

* **Nishad PM** M.Sc., M.Phil. Seven months Worked as a project trainee in Wipro in 2005, five years experience in teaching, one and half year in JNASC and Three and half year in MES Mampad College. He has published eight papers national level/international conference and journals. He has presented three seminars at national Level. Now he is pursuing Ph.D Computer Science in Dr. Mahalingam center for research and development at NGM College Pollachi.

** **Dr. R.Manicka chezian** received his M.Sc., degree in Applied Science from P.S.G College of Technology, Coimbatore, India in 1987. He completed his M.S. degree in

© 2012, IJARCSSE All Rights Reserved

Software Systems from Birla Institute of Technology and Science, Pilani, Rajasthan, India and Ph D degree in Computer Science from School of Computer Science and Engineering, Bharathiar University, Coimbatore, India. He served as a Faculty of Maths and Computer Applications at P.S.G College of Technology, Coimbatore from 1987 to 1989. Presently, he has been working as an Associate Professor of Computer Science in N G M College (Autonomous), Pollachi under Bharathiar University, Coimbatore, India since 1989. He has published thirty papers in international/national journal and conferences: He is a recipient of many awards like Desha Mithra Award and Best Paper Award. His research focuses on Network Databases, Data Mining, Distributed Computing, Data Compression, Mobile Computing, Real Time Systems and Bio-Informatics.

© 2012, IJARCSSE All Rights Reserved